University of Calgary

# CPSC 319 Data Structures, Algorithms, and Their Applications

## - Assignment 3 -

Name: Saina Ghasemian-Roudsari
UCID: 30113011

Tutorial Section: T09

## Complexity Analysis

1. Assuming that the records are inserted into the tree in random order, what is the height of your tree expressed using big-O notation?
   When assuming that records are inserted in random order, each node can either be inserted into its respective position in the left or right subtree of the root node. All nodes in any left subtree will be less than or equal to the parent node, and any nodes in the right subtree will always be greater than the parent node. Considering the height of the entire tree to be represented by k (from the root to the furthest node), and n being the total number of records stored in the data structure. As covered in lectures we know that the height (k) also is the number of levels in the binary search tree. So, since the maximum number of nodes per level is 2^k nodes per level, we can see that the sequence is n = 2^0+2^1+2^2+2^3+...+2^k, which can also be written as n = 2^(k+1)-1. Rearranging this equation since we are solving for k we get k = log2(n-1)-1, which means the height of the tree(k) in big-O notation is O(logn).

2. What is the worst-case height of the tree? What input gives the worst case?
   The worst-case height of the tree is when the binary search tree degenerates into a linked list, which makes it either fully skewed to the left side of the tree, or fully to the right side of the tree. If this were to happen the height of the tree would then be n-1 (n is the total number of nodes stored in the binary search tree). The height of the binary search tree in the worst-case scenario can be expressed in big-O notation as being O(n).
   The input that would give the worst case is when the records(nodes) being interested in the tree are already sorted prior in either ascending or descending order. If already sorted in ascending order the tree will be skewed to the right because the first record being inserted is the smallest. If the input is in descending order the tree will be skewed left because the first record inserted is largest (opposite of ascending). So, again the height of the binary search tree in worst-case will be O(n).

3. What is the worst-case space complexity of the depth-first, in-order traversal and breadth-first traversal? Compare your implementation of these two methods: is there one that will outperform another in terms of memory usage for a specific data set?
   The worst-case space complexity of the depth-first (which is recursive), in-order traversal depends on the maximum height of the tree which means its n-1 (where n is the number of nodes in the tree). This is expressed as O(n) and happens when the binary search tree is unbalanced into a linked list, and is using memory on the stack to store all of the nodes. The breadth-first traversal method uses a queue to store and write each node's information in

the correct order. Using a queue, for the worst-case, we might need to store every node's information before dequeuing and writing to the file. This results in a space complexity of O(n). Comparing both algorithms they have the same complexity, breadth first will be better with using a binary search tree that has degenerated into the linked list.

## Binary Search Tree